Module Interconnection Frameworks for a Real-Time Spreadsheet

Progress Report
May 30, 1993

Program: SBIR N92-112

Scientific Officer: Ralph Wachter, Naval Research Laboratories

Principal Investigator: Richard Clarke, RTware, Inc.
714 9th St Durham NC 27705 919/286-3114

Overview

The first month of this work has been very fruitful, yielding a detailed understanding of how MIF can be used to implement specific features of a real-time distributed spreadsheet. A number of ideas will be presented in this report. These ideas envision a system which combines the advantages of high-level system design with the very rapid implementation/prototyping capabilities already demonstrated in numerous applications by the ControlCalc system. In particular, a MIF system solves a number of problems of distributed programming, and of integrating services provided by the spreadsheet into a large scale modular system implemented in non-spreadsheet languages.

The focus of the investigation so far has been Polylith, the software bus system developed at the University of Maryland under Jim Purtilo. I have downloaded all the currently available Polylith source code and documentation and built it on our UNIX systems. I have also reviewed in detail the MetaH system documentation provided by Steve Vestal of Honeywell. Attending the Prototech conference in May provided a clear view of how our project fits into the overall Prototech charter. Some off-line discussions at the conference were also illuminating. Finally, discussions and meetings with Paul Hagger of UMD have helped focus our thinking on some particular uses of Polylith in the near term.

In addition to specific work on MIF, RTware has been working on a proposal for one of our customers for a function block diagram system based on ControlCalc. This work can both benefit from a MIF system, and shows significant parallels to the MetaH system at Honeywell and other high-level diagramming systems. For reference, a draft of the summary proposal is attached. While this proposal does not specifically address MIF technology, we are currently evaluating Polylith's ability to provide the framework for the distributed aspects of that proposal.

Polylith versus MetaH

Based on preliminary indications from Steve Vestal, I did evaluate the MetaH system for potential use as a real-time module interconnect framework. I have decided to use Polylith because, as far as I can see, MetaH is not a generic MIF system. Rather it is a tightly integrated application language which focuses on structural design and analysis of cooperating process modules. It is based on some specific tools, including a kernel, developed by Honeywell, and is tightly coupled to Ada. Polylith, on the other hand, provides simply an interconnection mechanism, without regard to the method used to develop either the modules or specify the high-level system. MetaH seems to provide such capabilities only internally, if at all, without opening them up to outside modules.

MetaH is very interesting in as much as its diagram programming system is analogous to the system RTware intends to create. However, MetaH is not unique in that regard. Systems such Hewlett Packard's Visual Engineering Environment (VEE), Labview, Genesis and Digital Equipment Corporation's laboratory instrumentation system are all similar in their function block programming. MetaH does provide some real-time semantics, similar to those RTware is proposing.

Polylith as it is available today has two major limitations: static configuration and no shared memory capability. Work at UMD (for example White, 1992) is addressing these limitations. In the meantime, these limitations do not actually impede our efforts. All of the control systems developed with ControlCalc are static, fully compiled configurations. Even when distributed, control applications will generally have all the possible modules and nodes fully specified when the system is started. Shared memory versus message passing features would be useful, but the distinction can be handled at the compiler level at this time. The issue such as using caching schemes to reflect shared memory over networks requires further investigation, but is, I believe, more of a performance issue rather than a semantic one.

Specific Features Proposed

I have come up with a number of specific features and capabilities which will benefit from the Polylith MIF. Each is discussed in some detail below. They are:

1) External access to spreadsheet services.

2) Calling remote modules from within the spreadsheet.

3) Establishing data stream connections between spreadsheet instances.

4) Distributing executable code segments among processors.

1) External Access to Spreadsheet Services.

The first feature would allow external processes to access the spreadsheet through a procedural interface. This would be a new capability for spreadsheets in general, since spreadsheets normally do not provide the ability to construct callable procedures internally, let alone for external use. We do not propose to somehow make the spreadsheet a procedural language. Instead we propose to allow the spreadsheet user to specify procedural interfaces that can a) access data in the spreadsheet and b) trigger recalculation within the spreadsheet.

The user will be able to define specific, named procedural interfaces which include input and output parameters and trigger mode operations. In specifying the parameters, the spreadsheet user will specify which data cell in the spreadsheet corresponds to each parameter. The basic procedure for responding to a procedural request would be to copy input parameters into their associated cells, trigger a recalculation event, wait for a completion event and copy output parameter from their cells. Trigger and completion events would be optional. Whether they are used would be part of the procedure definition, not a run-time option for the caller. This simple procedural interface will support the most common operations done against real-time control systems: setting control system parameters and reading the control system variables such as sensor input values. The trigger event option allows the external processes to control evaluation of the control system at a high level, and also allows the spreadsheet to act as a computation engine for other processes. Note that the initial implementation of this interface specifically will not support procedural calls within a process. That is, it will not allow spreadsheet code segments to be linked into other modules. Instead it will be strictly an interprocess communication mechanism.

Under Polylith, these interfaces are specified application specification language and a remote process is started for each interface. These processes may access the same spreadsheet, or different spreadsheets. Run-time errors will result if the specified service is not available from the particular spreadsheet.

2) Calling remote modules from within the spreadsheet.

Simply put, this is a flexible way of adding functionality to the spreadsheet. Currently users can compile in their own 'C' functions for execution within any ControlCalc process. By using the Polylith system, callable subroutines would correspond either to simple subroutines or to remote procedure calls. This would allow subroutines to be implemented in any language that supports the Polylith interface. It will also allow spreadsheets to access the external functions exported by other spreadsheets. This approach would be a useful generalization of message passing functions previously considered by RTware to support distributed applications.

3) Establishing data stream connections between spreadsheet instances.

Currently, ControlCalc allows spreadsheet FIFO cells to be connected through stream sockets using TCP/IP. We intend to host this capability on Polylith in order to provide independence from the communication mechanism.

4) Distributing executable code segments among processors.

A major design goal of the MIF project for ControlCalc is to allow executable code segments to run on distributed processors. In this scenario, there would still be one master shared memory spreadsheet, with the various "scans" acting as separate threads of control. These scans, however, could be instructed to execute on a remote processing unit that did not necessarily have direct shared memory access to the spreadsheet. The ControlCalc compiler would therefore generate cross reference data to allow reflection of variables across the network connecting these segments. The code generated to reference externally visible variables could invoke a cache control routine which posts updates to the network only when required. We do need to analyze the performance and determinism effects of the caching approach, comparing it to the more brute force method of periodically flushing external data. The periodic flushing approach has the advantage of being deterministic and of being independent from the scan code. Synchronization operations with signals and fifos would of course require immediate handling. While Polylith does not address this shared memory problem, this is an active area of research at UMD.

Specific Actions Undertaken

At this point we are proceeding to implement the first proposal using Polylith. External access from non-ControlCalc applications is a feature which will have immediate benefits, and one for which a standard interconnection scheme such as Polylith is required. The immediate benefit is the ability to integrate ControlCalc applications into a larger system, without having to scrap existing designs and modules. Researchers and system developers can then experiment with ControlCalc's abilities rather easily. Possible modules that could be implemented with ControlCalc include

1) Front end I/O processor.

2) Local high-speed feedback controller.

3) Operator input and display module.

4) Report generator and recorder.

5) Logic and Function solving engine.

As we become more familiar with Polylith, we will move our history cell stream connections over to Polylith from their current socket implementation.

Phase Two Proposal

At this point, we believe that DoD will realize significant benefits in software productivity with the MIF ControlCalc system (see benefits discussion below). Our work on Phase I will include demonstratable uses of Polylith, in particular the ability to invoke spreadsheet services and attach to spreadsheet data streams from another language (C in our case). For phase II, the attached proposal for a graphical functional block diagram system is a major component. As written, the proposal does not address MIF. The benefit of adding MIF to that system would be to make it capable of handling heterogeneous modules, and providing a platform for the distributed features described.

The heterogeneous module capability of MIF is critical to making high-level programming or prototyping systems open and extensible. If, for example, MetaH were implemented on top of Polylith, then ControlCalc modules could be included in their system, and vice versa. The actual high-level design language would not be tightly coupled to the definition of modules and interconnections, as they are now. RTware, for instance, is considering an interim script language for building function block-based system before implementing the graphical editor. This could be used to develop the MIF module system, with much less programming effort than a graphical editor would require. A script language could also be more easily generated by database programs, allowing auto-configuration of systems. Once running under a script language, the same modular design would work under a graphical system.

Another area to be addressed in Phase II would be the addition of timing analysis to ControlCalc. This would provide approximate timing ranges for execution of a module created with ControlCalc. Currently, ControlCalc does not provide timing analysis and performance prediction tools. It does run with a real-time operating system with guaranteed deterministic response. Discussions with Jane Liu at the Prototech conference indicate that the system can be formally described as rate monotonic. Dr. Liu's work on predicting real-time performance is well advanced, particularly for rate monotonic systems, even to the point of presenting a close to commercial grade X-Windows interface. We propose to use the tool as an analysis utility for predicting schedulability based on the timing estimates and scheduling constraints from any given ControlCalc application.

Benefits to DoD

The benefits to Department of Defense software development work are expected to be software productivity. Productivity improvements are derived in two ways. First, the ControlCalc spreadsheet programming language allows a very simple (too simple, to many C programmers) way of expressing functional programs. Functional programming itself is well suited to control programming, which often consists of continuous solving of control laws on a deterministic schedule. Second, the MIF interface allows the modularization of function blocks that are created in the spreadsheet. This results in reusability and in reliability, since each module has a defined interface and can be independently tested and verified. Modularity also means that spreadsheet programming can be integrated into larger frameworks on a step by step basis, and can be limited to those areas of the application where its paradigm is appropriate. This greatly reduces the risk of moving to a new language.

RTware does not have scientific data to support the claim of improved productivity, as we do not have the resources to perform parallel implementations of our customer's applications. However, rather dramatic results are readily apparent by looking at casc studies. Two of these cases involve applications directly relevant to the Navy.

In the first application, ControlCalc was used for a Navy airfield lighting system. The application involved controlling the runway light banks and monitoring relays to detect output faults. Monitoring was done remotely, using radio modem links to an optomux-compatible I/O network. The user interface was based on a graphical picture of the runways, with dynamic colors representing different states. Constraints were applied to operator actions through rules defined for different classes of lighting. Pop-up control panels are used to let the operator monitor and adjust lighting parameters. Dynamic color control is provided through tables of data in the spreadsheet, with a color mixing option for customization. This work was performed in about four months by a single engineer with limited software experience. It was successfully reviewed by the Navy and installed in Beaufort South Carolina in April, 1993. Not incidently, the same project was attempted a few years before using third-generation programming techniques, but was cancelled due to severe cost and time overruns.

In the second application, Encore Computer Corporation has produced a system health and performance monitoring and control system for their Infinity line of parallel processing Unix computers. The application required porting ControlCalc to the 88K processor, to the UNIX operating system and designing and implementing an X-Windows control panel interface builder. About ninth months after first delivery of an alpha-testable port, the application is now being released as a standard, embedded feature of the Infinity Operating System, under the trade name CommandCenter. Reviews by mainframe users and analysts indicate that the application provides state of the art visualization and control of the systems operating condition. It consists of 95 different graphic screens, about 1.5 MB of compiled ControlCalc code, 600 I/O points, extensive data logging and analysis, rule solving and operator actions. The system logs reports, sends mail, supports multiple logins, does continuous data acquisition and still takes less than one percent of system processing time. The entire application was developed by a single project engineer, with much of this time spent qualifying the porting and development work required. He estimates that the actual application development work, including I/O drivers, took about six months. This application is relevant to DoD because Encore has a number of contracts with DoD for mainframe replacments, including an order from NRL. The MIF work we are doing is also very relevant because the next phase of this project is to make the system fully distributed using Encore's Reflective Memory technology.

A final application is not for the Navy, but involves control of turbomachinery, of the type typically used on large Naval vessels. Just three months ago, a major turbomachinery controls manufacturer started using ControlCalc to design their new generation low-cost control product. After starting work on March 1, 1993 they had the system in alpha test on May 1, 1993. No problems were found in alpha test, so they are now starting to market it. The general manager of the company informed RTware that, if anything, we undersold the productivity improvements from ControlCalc. Their application consists of a full authority, multi-loop controller with a complete graphical user interface for monitoring and adjusting system parameters.

It is interesting to note that the turbomachinery company's software engineering staff did not believe that a spreadsheet programming paradigm would be able to do the job at all, let alone in three months. In this case, one control engineer simply did it. This resistance to a much simpler programming method is typical of software engineers, as we saw in some discussions at the Prototech conference. It is however, simply wrongheaded. We have found that non-programmers can produce highly complex, reliable control systems even without the benefit of modular programming techniques. This is the experience of spreadsheet users in general, which is why spreadsheets are one of the top three applications in use today.

Conclusion: The integration of a spreadsheet programming paradigm into a modular, function block system will provide very significant productivity gains across a wide range of applications and allow professionals with domain-specific expertise to directly implement their applications.

Graphical Programming
with
ControlCalc

A System Design Proposal

Introduction:

RTware proposes to enhance its ControlCalc product to provide a fully graphical programming interface. The goal of these enhancements is a system which will provide:

1) The flexibility of spreadsheet programming for designing function blocks.

2) The speed of the spreadsheet compiler.

3) Hierarchical Sequential Function Chart Programming.

4) Graphical Panels for I/O and scan configuration.

5) Self-hosted and remote downloadable execution.

The following two major programming tasks are required to achieve the first four goals..

1) Producing a graphical spreadsheet interface

2) Producing a function block diagram editor.

RTware is currently working on a graphical version of the Controlcalc spreadsheet. This task is well-defined, since graphical spreadsheets are common in the commercial world. The first version of this system will run under G-Windows, providing a self-hosted real-time environment. The next versions will run under X-windows and MS-Windows. Both systems will run with real-time kernels running under the window system. MS-windows will run either with the OS-9000 or iRMX operating systems (from Microware Systems Corp. and Intel Corp. respectively). X-windows can run with a variety of real-time UNIX or UNIX-compatible operating systems, including LynxOS and OS-9.

The second task is a significant programming effort. However, ControlCalc's range save and range load features provides a simple way of implementing function blocks. The main body of this paper discusses some of the main design ideas for this system.

The fifth goal requires a medium level enhancement to the ControlCalc compilers, and careful choice of the distributed environment. This work is orthogonal to the other tasks, since it involves only the compiled control code produced and does not affect the user interface. The major issues involved in remote execution are the mirroring of shared memory over the network and the separation of run-time data from the development system.

Benefits:

Before proceeding with such an ambitious software development project, there must be sufficient benefits to the resulting product to justify the development cost. The entire system is designed to be an application development environment, and the following benefits are expected:

1) Reduced initial development cost.

2) Lower life-cycle software cost.

3) Increased reliability.

4) Reduced hardware costs.

5) Higher performance.

Reduced initial development and lower life-cycle costs are based on the simple programmability at the low level provided by the spreadsheet interface combined with the high-level graphical view of the system provided by the function block diagram system. RTware has already observed approximately 75% reductions in application development times by the users of the ControlCalc spreadsheet alone. Furthermore, these users have a wide range of technical backgrounds and skill levels, usually not in computer science.

The function block diagram system allows ControlCalc productivity to be encapsulated and reused in a modular fashion. Users of the resulting system will be able to develop libraries of function blocks which are both totally modular and easily enhanced. This capability will overcome a primary limitation of function block programming systems, the lack of an easy end-user programming method for creating and modifying function blocks.

Increased application reliability is expected, based on the understandability of the flat programming model of the spreadsheet and its interactive nature. The spreadsheet programming model is essentially a sequential function evaluation system, with each function's result available on-line in a simple visual matrix. The highly interactive nature of the system allows testing and inspection of the application to be done at all points in the software development cycle. Simulation, prototyping and experimentation with the application require no additional tools and can be done on the actual application. The typical ControlCalc development cycle involves iterative, bottom-up design, with continuous testing. The fact that functions are immediately active and results are always on-line makes most programming errors readily apparent. To date, ControlCalc users have not once found it necessary to resort to the traditional debuggers available for testing the run-time code.

Reduced hardware costs and higher performance are expected. This is not due to any particular performance enhancement, but by comparison with other end-user programmable systems. It is axiomatic that second and third-generation programming languages are capable of achieving the maximum performance allowed by the system platform (hardware and operating system). However, in the real world excess hardware costs are often incurred to achieve higher programming productivity. A simple example is the common use of programmable controllers with separate computers for both user interface and supervisory control. In many applications the final product is actually monolithic, not distributed, so this architecture is not determined by the application. The architecture is determined by the software tools available, i.e. ladder logic programming and GUI-builders. In the ideal situation, decisions on the extent and configuration of distributed software would be made by analyzing the application requirements, instead of being constrained by the tools available. The key feature required for this qualitative advance is a fast, embedded compiler that imposes few, if any, constraints or delays on the application developer.

Range Files as Function Blocks

The fundamental function block design will be based on the existing "range file" capabilities of ControlCalc. This capability allows users to save any rectangular range of the spreadsheet to a file. The file contains all the cells in the range and the names of all cells in that range. The range can later be loaded into another spreadsheet. When loading, the user specifies the top left corner location of the range. The loaded range has the same shape as the original range. All relocatable cell references are adjusted to reflect the new location and the range's names are added to the spreadsheet's tag name table.

This simple function has a number of limitations that will need to be changed to support true function block programming.

Enhancements required for range files

The existing range file capability allows only one copy of a range file to be loaded into a template if that range file contains any symbolic names. If the same range file is loaded again, there will duplicate name conflicts. One solution to this problem is to load a range file by specifying a new location and a name for that instance of the range in the spreadsheet. By doing this, the internal tag names can be concatenated with the instance name to maintain uniqueness. This solution makes the naming private to each instance of the range. It would also be possible for external functions to reference these internal variables, but that would not normally be done within a function block system.

External references to the range file function blocks currently must be done manually. However, the relocatable nature of the block's cell references does allow some shortcuts. For example, the range file could have a reference to a cell one column to the left of the block in a particular row. When that range file is loaded, that reference would relocate and refer to whatever cell is in that relative location in the new spreadsheet. By doing this, and by avoiding the use of symbolic names in the range files, users currently can paste range files in repeatedly, and create tables of parameters in such relative locations. This method is not sufficient for a true function block system, which would require references between blocks to be resolved by the user drawing lines between attachment points on different blocks.

To support this dynamic linking between function blocks, we propose to add a new type of cell called a "link" cell. A link cell will be a cell which can contain only the name of another cell. It would be typed just like expression cells, and could be either empty or contain the text name of another cell. Expressions could reference link cells normally. When the spreadsheet is evaluated or compiled, references to link cells would resolve to references to the cell whose name is in the link cell. Of course, unresolved or improperly typed references would result in syntax errors. Note that link cells are needed only for external references from the range block. They provide a way of keeping all relocatable references inside the function block, and a place to record the resolution of the essentially indirect references. Link cells would involve no overhead in compiled run mode, since the link would be resolved at compile time. We do not envision a need for dynamically changing links while the system is running. A manual spreadsheet command would be available for resolving link cells by editing after the range block is loaded. In a function block diagram system, the link cells would correspond to input points on the block.

References into a function block from the outside can be accomplished by simply adding the attribute of external visibility to a cell. This would not actually be necessary if links were established manually within the spreadsheet. However, when range blocks are used in a function block diagram system, the creator of the function block needs to be able to specify which internal cells are visible externally. These externally visible cells would correspond to output points in a function block diagram.

Overview of usage

The system will be used in a two part process: creating function blocks and using a function block diagram editor.

Function blocks will be created inside the standard ControlCalc spreadsheet. The creator must follow the rule that all externals references are made through link cells and must set the external visibility attribute where desired. The range can then be saved to a range file in the normal manner. A utility will be provided to allow the creator to specify an image piece and/or text name which will show up in the actual function block icons. Other decorative attributes of the block such as its shape and the position of the attachment points could also be specified with this utility. A default shape and position of the attachment points would be provided. Once created, the spreadsheet portion could be reloaded, edited and saved with its attributes.

Function blocks also have to be configurable. A set of parameters must be made available to the user, not as attachments, but for setting variables inside the block that are used by the expressions inside the block. These variables are entirely defined by the function block creator, and can be any type of cell. ControlCalc already has the ability to present a spreadsheet-based form(s) to the user in operator mode. This mode allows the user to cursor between pre-determined setpoint fields and change those fields. ControlCalc also supports the attachment of graphic windows to spreadsheet cells. These graphic panels can be used for changing or viewing data in the spreadsheet using a rich set of graphics gadgets. Either method could be used within the function block diagram system. In both cases, there are potentially two displays required, an edit-option display and a run-time parameter adjust display. The function block creator will be able to attach graphic windows to the block and indicate which window is shown initially under the twc conditions. The macro command cell capability already allows windows to be dynamically called up and removed by the end user. The creator simply needs to indicate which macro cells are to be initially executed. The rest of the interaction is entirely determined by the function block's contents. If the simple spreadsheet interface is desired, a mode setting could be used to determine whether the indicated cells are macros to be executed, or the top left settings of a spreadsheet region to be displayed in operator mode. Note: this determination could be made automatically by stating that if the cell is a macro cell, then it is executed, otherwise it is the top left corner.

Function blocks would be used with a function block diagram editor. This discussion will not cover the user-interface details of the graphical editor, as such systems are common and well understood. A general discussion and certain functional details are important, however.

The function block diagram user will place icons representing blocks in the diagram and connect reference points. Different symbols will represent the different types of reference points, and the editor will complain if illegal connections are made. The user would also be able to open up the function block and set parameters using either the attached window(s) or spreadsheet operator mode.

The function block diagram editor should be hierarchical, so that function icons can be grouped together dynamically and viewed as a single icon. Such icons would have to have a pre-determined look, but present a dynamic set of attachments. The icons in a group would then be edited on theit own display, and attachments outside the group would be indicated by attaching a border region or decoration of that display area.

Sequence of operations

The most difficult design area for the system is sequence of operations. Most function block diagram systems are really data flow diagrams (as in LabView). These systems schedule functions as data becomes available. This approach can result in unnecessary overhead and can also be non-deterministic, both of which are problems for real-time control systems. The data-driven approach requires that each block be separately scheduled and that an executive monitors data sources. While this is appropriate in some cases, in many applications the function blocks are best executed in a synchronous sequential manner. This is particularly true in feedback control systems. In these systems, such functions as input, scaling, limit checking, feedback equations and output are all inherently synchronized and run at a rate specified by the user. The obvious case for data-driven evaluation is in message-passing or networked applications where data is processed, possibly asychronously, only when a message or new data is available. Distributed control systems are the obvious example.

ControlCalc itself already allows both methods to be implemented. Multiple scheduled code segments are provided since each page of the spreadsheet is a separate control task. Synchronization is provided through the signaled scan option, and message passing or data piping through history (FIFO) cells. History cells can optionally be configured so that read accesses wait for data and write accesses wait for room, and with locking to ensure exclusive access. Monitor-style control can be constructed using the signal and wait primitives. A network-based I/O option also allows data to be piped between history cells on different machines using streams. On the other hand, within each page execution is strictly sequential (disregarding GOTO's). Depending on its requirements, an application can be as simple as a single scan doing one continuously iterating control loop or a whole set of cooperating scans scheduling themselves as necessary.

Sequential Evaluation

Within a function block diagram system some method has to be provided that lets the user take advantage of both types of scheduling. The simplest method is sequential evaluation. This would allow the user to specify that the entire diagram, or a group within the diagram is executed sequentially. Sequential evaluation would probably be the default mode. The system would use built-in rules based on examining connections to attempt to determine the proper sequence of evaluation. The user would be able to override those rules by drawing explicit ordering links between blocks. Circularities in the ordering links would be forbidden.

This results in a system which is static and sequential in its evaluation order. The underlying method for evaluating the function blocks would actually be the ControlCalc spreadsheet compiler. Each block would be placed in an arbitrary location in the spreadsheet, which then provides the means for recording the linkages the user creates. The compiler would compile the blocks not in the standard row-major order of the spreadsheet, but block by block according to their ordering. This would be an elaboration of the "one page is one task" structure now being used by ControlCalc. Instead, diagram information would be used to drive the compilation and actual spreadsheet layout would not be important.

Multi-tasking evaluation

The user would be able to define multiple groups that could be separately scheduled. References between groups would still be allowed. Currently, multiple tasks are allowed on one machine, with all pages sharing the three-dimensional spreadsheet as a sort of shared memory data base. This approach would still work with the diagrams. All groups would actually be placed in the global spreadsheet, so data references would resolve automatically. The compiler could also be used to handle distributed applications, where groups are evaluating on different machines. Inter-group references would compile to message-passing calls rather than simple data references.

Event or data-driven evaluation

Event-driven evaluation is already supported by ControlCalc. Currently a page can be set up as a signaled scan, which means it evaluates once each time it receives a signal (event). A signal function is provided to allow spreadsheet expressions to send signals. Also, a wait function is provided to allow scans to synchronize at any point in the code. This capability is implemented using global events, either named events under OS-9/OS-9000 or System V IPC semaphores under UNIX.

Under a function block system, event scanning could be declared as an evaluation mode for a sequential group, instead of giving the group an iteration rate. This is equivalent to the method currently used with ControlCalc pages. Alternatively, a graphical method could be used, involving a standard function block that represents the global event. That icon could be connected to multiple signal senders and one receiver. Linking a cell to a signal send would require that references to that link cell use it as the signal i.d. number, and vice-versa. Sent signals would also be able to be linked to a schedulable group to trigger its evaluation, using a special attachment decoration on the icon. Timer functions could be used in this manner to graphically illustrate the source of iterative evaluation, since the difference between iterative and event-driven for the groups is simply one of how the evaluation is triggered. It is also possible to implement monitor functions (lock and unlock) which can provide exclusive control among any group of functions using the same monitor. The monitor icon would be slightly different from a signal icon, since it is initialized in a different state.

Finally, some enhancements can be made to the history cells to allow evaluation triggers to be set by a history cell reaching a certain threshold. This would require that functions that write to history cells check the threshold and send a signal when the threshold is exceeded. The history cell would have to maintain some state information which allowed another function to reset the signal so multiple signals are not sent for the same threshold condition. In general, there would have to be a history link cell which allowed a function block to externally reference a history buffer, but conceptually this would work the same way as simple data cells. A history cell configured to send signals on reaching a threshold would then result in an interface decoration on the function block that could be attached to a global event icon as a sender. With the locking option on history cells, this approach could be used to create completely data-driven systems. Using network drivers to stream history cells together would then allow a diagram to span distributed systems.

Graphical Operator Interface

While a function block diagram is a useful tool for a system designer, in production most applications should present the simplest possible control panel interface to the operator. ControlCalc already allows control panel graphics to be created and attached to cells in the spreadsheet. Each function block can be configured with its own control panel window for run-time use. However, there has to be some way of creating a master control panel which can be attached to either the local function block control panels or to other global control panels for applications which don't need to involve the operator in the diagram system. The simplest solution to this requirement seems to be letting the user create a custom function block and link it to external data points in the usual manner. The ability to have a spreadsheet interface on-line for creating custom function blocks is crucial for this capability. It is likely that every application will have semi-custom set of control panels, using the standard function block panels in some cases and not in others. It may also be useful to allow command macros to include sub-panels from function blocks, thereby allowing access to private data. This lets the operator, for example, select a PID control block and adjust its internal run-time parameters without having to export the entire parameter set from the block. Including these sub-panels would then allow windows with banks of sub-panels, similar to the modular design of hardware control panels throughout industry.

Summary

The result of this design should be a system which provides the high-level graphical interface to a control or data acquisition system without sacrificing the performance and programmability already achieved by ControlCalc. The option of both event-driven and sequential, shared variable scheduling allows the function block system to cover a much broader range of applications than existing systems. The use of a spreadsheet for creating function blocks will speed the creation of function blocks and allow users to easily create function block libraries and to create those custom features needed by particular applications. Finally, a function block system is inherently suited for the creation of large scale distributed systems, where an overall understanding of the system architecture is needed.